VALPAR
INTERNATIONAL
3801 E. 34TH STREET
TUCSON, ARIZONA 85713
602-790-7141

valFORTH T.M.
SOFTWARE SYSTEM
for ATARI*

DISPLAY FORMATTER

*Atari is a trademark of Atari, Inc., a division of Warner Communications.

valFORTH T.M.

DISPLAY FORMATTER

Version 1.1
March 1982

The following is a description of commands used in creating video display lists on the Atari 400/800 series microcomputers. Creating custom display lists allows for innovative graphic layouts of games, simulations, or business applications which utilize both hi-resolution graphics and text simultaneously.

valFORTH™
SOFTWARE SYSTEM

DISPLAY FORMATTER

Stephen Maguire

VALPAR INTERNATIONAL

Disclaimer of Warranty
on Computer Programs

All Valpar International computer programs are distributed on an "as is" basis without warranty of any kind. The total risk as to the quality and performance of such programs is with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair.

Valpar International shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss, or damage caused directly or indirectly by computer programs sold by Valpar International. This disclaimer includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

Defective media (diskettes) will be replaced if diskette(s) is returned to Valpar International within 30 days of date of sale to user.

Defective media (diskettes) which is returned after the 30 day sale date will be replaced upon the receipt by Valpar of a $12.00 Replacement Fee.

An indepth explanation of display lists was written by Dave and Sandy Small in a series of articles found in Creative Computing. We suggest that this be read to get the most out of this valFORTH package.

STROLLING THROUGH THE DISPLAY FORMATTER

In Atari Basic there are many different graphic modes. Some of these are text modes, some are graphics modes, and some are mixed. These different graphic modes are based upon display lists. A display list is a list of display instructions which tell the video processor whether a particular portion of the screen is to be high resolution graphics or normal text. Any given section of the display can actually take on one of 18 different characteristics.

Let's take a look at the display list for a graphic 0 display: (These values are in base 16)

BC20	70	
	70	24 blank scan lines
	70	
	42	
	40	DM jump to BC40
	BC	
	2	
	2	
	.	
	.	23 graphic 0 lines
	.	
	2	
	2	
	41	
	20	jump to BC20
	BC	
BC40		start of display memory

Each opcode 70 instructs the video processor to display 8 blank scan lines. Opcode 2 produces one standard graphic 0 text line. Opcode 42 is a modified 2 instruction. In addition to creating a standard text line, it also informs the video processor where the display memory is located (the address is found in the next two bytes). At the end of the list there is a three byte jump instruction which transfers display list interpretation to the address specified in the next two bytes of the list. Each of the graphic settings have a similar list. This valFORTH package allows you to design your own lists. Let's make one now.

Look in the directory (screen 170) and load in the display formatter. Most of the formatter words begin with DB (for display block). To initialize the system type:

```
DBINIT
HEX
```

This initializes the system and puts it into the more useful hexadecimal mode. Graphic mode 8 is a high resolution graphic mode with a four line text window at the bottom of the display. Let's make a display with a four line text window at the top of the screen followed by the high resolution graphics plate. First, we need 24 blank scan lines at the top:

```
70 DBM
70 DBM
70 DBM
```

The DBM command stands for "Display-Block Make." It takes the opcode on top of the stack and tacks it onto the end of the display list currently being created. Additionally, it enters an address into the array DBLST which points to the first byte of memory used by that display block. There is a plural form of the DBM command:

```
3   70 DBMS
```

This adds 3 opcode 70's to the current display list. Now let's add the four line text window. Recall that a normal text line has an opcode of two:

```
4   2 DBMS
```

Note that the display memory jump described earlier is automatically inserted into the display list. Now we need to define the high resolution portion of the display. A standard graphic 8 line has an opcode of $F (15 in decimal). Let's create 20 graphic 8 lines (20 in base 16 is 14).

```
14   F DBMS
```

This list is good enough for now. To verify that it has been entered properly, type:

```
DMPLST
```

You should get something like:

BLK	ADDR	BYTE	MODS
0	A100	70	
1	A100	70	
2	A100	70	
3	A100	2	J A100
4	A128	2	
5	A150	2	
6	A178	2	
7	A1A0	F	
8	A1C8	F	
9	A1F0	F	
A	A218	F	
B	A240	F	
C	A268	F	
D	A290	F	
E	A2B8	F	
F	A2E0	F	
10	A308	F	
11	A330	F	
12	A358	F	
13	A380	F	
14	A3A8	F	
15	A3D0	F	
16	A3F8	F	
17	A420	F	
18	A448	F	
19	A470	F	
1A	A498	F	

Note the automatic insertion of the display memory jump in block three. Display memory cannot cross a 4K memory boundary without a display memory jump. As each display block is added, a check is made to detect any 4K memory crossings caused by the display block. If the block does cross, a display memory jump is automatically inserted into the list to account for it.

Now that we have a display list, let's enable it. There are several ways to activate a list. For now type:

```
MIXED  CLS
```

This MIXED command enables the new display list and also re-directs output to the display memory specified by the list. This allows for interactive display list creation. There should be a recognizably different display. Hold down the RETURN key and watch how the "ok" message is displayed as the cursor

moves down the screen. You should see "ok"'s on the text lines, but in the high resolution lines, it shoud look quite different. You can type in a high resolution mode because the Atari operating system does not know that the display list has been changed. To return to a normal display, the GR. command is used:

```
0 GR.
```

Dump the display list again using the DMPLST command. Let's put some text lines in at block B. To do this type:

```
     B DBPTR
10   2 DBMS
```

The DBPTR command positions the display list pointer to the specified block. That block then becomes the end of the list. After that, we add 16 (10 hex) graphic 0 lines. Dump the list again and verify that this is indeed what was accomplished. To view this new display, type:

```
MIXED  CLS
```

Hold down the RETURN key again. Notice what happens as the cursor passes through the high resolution section and then back into the second text section. Type DMPLST again while in this mode and notice that everything works the same, the data is simply displayed differently. To get out, type "0 GR.".

Besides adding display blocks onto the end of a display list, the display formatter allows display blocks to be inserted and deleted as well. Block two has an opcode 70 which produces 8 blank scan lines on the video screen. By deleting this block from the list, the entire display will shift upwards by 8 lines. This is accomplished using the DBDEL command:

```
2 DBDEL
```

Dump the list and verify that the block has indeed been deleted. Enable the list using "MIXED CLS". Note that the first text line appears much higher than usual on the video screen. While still in this display, execute:

```
4  6 DBDELS
```

This will delete the four display blocks starting at block six. In this case, the four high resolution display lines are deleted. Type "MIXED CLS" and watch the screen shrink slightly as the display blocks are extracted.

Display blocks can be inserted using the DBIN command. When a DBIN command is executed, the specified opcode is inserted into the specified block. The opcode previously in that block and all opcodes following are pushed back by one block. As an example, we will insert opcode 70 (8 blank scan lines) at block five. This will do it:

```
70 5 DBIN
```

"MIXED CLS" will activate the new list. Press the RETURN key a few times and notice how the output routines seem to ignore the blank scan lines. The DBINS command is a plural form of the DBIN command. Let's insert a different opcode other than 2 or $F. Opcode 6 is a mode which displays colored characters which are much larger than normal. This will insert three opcode 6's at block 9:

```
3  6  9  DBINS
```

Activate this new list in the normal way and experiment with it. The following section describes all of the available opcodes. Experiment with these as you read about them and you should have no problem understanding any of them.

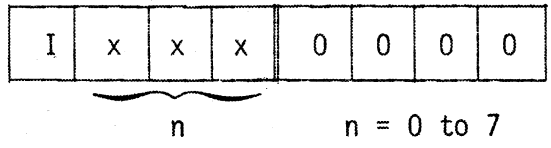
This brief explanation of display list formatting should show the power available to the programmer who wants to get that unique display. There are many more commands available for use. These are explained thoroughly in the glossary at the end of the next section.

DISPLAY LIST INSTRUCTIONS

There are four basic display list instructions. Those that produce blank scan lines, the display list jump, the jump on vertical blank, and the display block instructions. This is a description of these four basic instructions.

Blank Scan Lines

Byte form:



This opcode produces n+1 blank scan lines of color BAK. No video memory is used by this instruction.

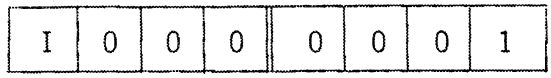
If the I bit is set, a display list interrupt (DLI) will occur upon interpretation by Antic (the video processor).

The 8 legal values are:

$00 =	0	1 blank scan line (128 with I bit set)
$10 =	16	2 blank scan lines (144)
$20 =	32	3 blank scan lines (160)
$30 =	48	4 blank scan lines (176)
$40 =	64	5 blank scan lines (192)
$50 =	80	6 blank scan lines (208)
$60 =	96	7 blank scan lines (224)
$70 =	112	8 blank scan lines (240)

Display List Jump

Byte form:



This command instructs Antic to search for the next display list instruction specified by the address contained in the next two bytes of the display list. The low byte of the address is found lower in memory. This command is used primarily to continue a display list across a 1K memory boundary (Antic will not handle this properly). This is the only instruction not supported by the display formatter since its occurrence is rare. It is explained here for completeness sake and its use is absolutely forbidden. Future releases may have this implemented.

If the I bit is set, a display list interrupt will occur upon interpretation by Antic.

Legal form:

$01	addr-low	addr-hi	Transfer display list interpretation to addr.

Jump On Vertical Blank

Byte form:

I	1	x	x	0	0	0	1

This three byte opcode instructs Antic to transfer display list interpretation to the address specified by the following two bytes (low byte of address first) and to pause until vertical blank occurs. Since display list processing halts, any remaining portion of the video display takes on the color of BAK. This command is not to be entered by the user. The display formatter automatically adds this to the end of the display list whenever it is moved or activated.

If the I bit is set, a display list interrupt will occur upon interpretation by Antic.

Legal form:

$41	addr-low	addr-hi	Transfer display list interpretation to addr. (65)

Display Block Opcodes

Byte form:

I	J	V	H	x	x	x	x
				n			

n = 2 to $F (15)

There are 14 display modes. Six are character modes, eight are graphic modes. Each of these modes varies greatly and will be discussed individually. But first, the four status bits I, J, V, and H, will be discussed as they function similarly for all display modes.

If the I bit is set, a display list interrupt will occur upon interpretation by Antic.

If set, the J bit instructs Antic to perform a display memory jump. Antic expects the next two bytes in the display list to point to the new display memory location. The first display block instruction should always have this bit set. Also, Antic cannot properly retrieve data from display memory across 4K boundaries. Thus, if the display memory must cross a 4K boundary, a display memory jump must be used. Note that the display formatter automatically takes care of these two problems for the user.

If set, the V bit informs Antic that the current display block is to be vertically scrolled upward according to the value in VSCROL (address $D405). Note that vertical scrolling is accomplished only if two or more consecutive display blocks have this bit set.

If set, the H bit informs Antic that the current display block is to be horizontally scrolled right according to the value in HSCROL (address $D404). Note that for horizontally scrolled display blocks, extra bytes of memory are needed. The exact number of bytes varies for different screen (playfield) widths. Use the following calculation:

```
# extra   =   X / n

where:  X = the number of characters/display block
        n = 4 for a narrow playfield
          = 5 for a standard playfield
```

There are no extra bytes for the wide playfield setting.

For example, a 40 character/line display block in the standard width would use a total of 40 + 40/5 or 48 characters. Note that only one of these extra bytes is actually used for the display.

The Character Modes

There are 6 character modes (opcodes 2 thru 7). All character modes work in the same way, i.e., the values in display memory are indices to a large "n" by 8 byte array. In some of these modes, the highest one or two bits are used to specify a color with only the remaining lower bits used for indexing. The following table gives information about each of the modes:

Antic mode	2	3	4	5	6	7
Basic mode	0	---	---	---	1	2
# color *	1.5	1.5	5	5	5	5
Chars/line narrow wid	32	32	32	32	16	16
Chars/line normal wid	40	40	40	40	20	20
Chars/line wide screen	48	48	48	48	24	24
Scan lines/ pixel	8	10	8	16	8	16
Bits/pixel	1	1	2	2	1	1
Color clocks per pixel	.5	.5	1	1	1	1

Colors:

mode 2: Takes the color of PF2 with the lum of PF1
(Artifacting/bleed very noticeable)
mode 3: Same as above
mode 4: Two bits/pixel in character definitions
00 = BAK 01 = PF0 10 = PF1
11 = PF2 if bit 7 of index = 0, else PF3
mode 5: Same as 4 above
mode 6: Most significant two bits of index
0 = PF0 1 = PF1 etc.
mode 7: Same as 6 above

The Graphic Modes

There are 8 graphic modes. Unlike character modes, the values in display memory are not indices into an array of character definitions, but rather are the definitions themselves. Depending on the graphic mode, these values give different results. The following table gives various information about each mode.

Antic mode	8	9	A	B	C	D	E	F*
Basic mode	3	4	5	6	---	7	---	8
# colors	4	2	4	2	2	4	4	1.5
bytes/line narrow wid	8	8	16	16	16	32	32	32
bytes/line normal wid	10	10	20	20	20	40	40	40
bytes/line wide screen	12	12	24	24	24	48	48	48
Pixels per normal wid	40	80	80	160	160	160	160	320
Scan lines/pixel	8	4	4	2	1	2	1	1
Bits/pixel	2	1	2	1	1	2	2	1
Color clocks per pixel	4	2	2	1	1	1	1	.5

*Mode F values differ when in GTIA modes

Colors:

```
mode 8:  Two bits/pixel, 4 pixels/byte
         00 = BAK     01 = PF0     10 = PF1     11 = PF2
mode 9:  One bit/pixel, 8 pixels/byte
         0 = BAK     1 = PF0
mode A:  Same as mode 8 above
mode B:  Same as mode 9 above
mode C:  Same as mode 9 above
mode D:  Same as mode 8 above
mode E:  Same as mode 8 above
mode F:  Take the color of PF2 and lum of PF1
         (if not in a GTIA mode)
```

GLOSSARY

(DBINIT) (dmem dlist ---)

The (DBINIT) routine initializes the display formatter. It expects two addresses on the stack. The address on top of the stack is used as the target address for the display list. The address found second on the stack is the target address for display memory. The display list is actually created in a c-array named DSPLST. Note that while building the list, no check is made to ensure that the display list does not cross a 1K memory boundary.

DBINIT (---)

Like the (DBINIT) command above, this initializes the display formatter. But unlike (DBINIT), this expects no arguments. Instead, these values are calculated automatically. The display memory address is top of memory minus 1F00 hex. This is enough for a full graphics 8 screen. The display list address is 256 bytes below the display memory address. Note that this is very memory wasteful, and should only be used while still learning the system. After that, (DBINIT) should be used.

DBPTR (block# ---)

This command instructs the display formatter to create the next display block in the specified "block#" of the current display list. To begin creating a new display list, use:

```
0 DBPTR
```

DBM (antic-mode ---)

The DBM command adds "antic-mode" to the end of the current display list. For example, to create a video display with a single line at the top of the screen, the following would be executed:

```
0  DBPTR     (new list)
2  DBM       (A graphic 0 line)
```

(Note: Antic mode 2 is a BASIC graphics 0 line.)

DBMS (#times antic-mode ---)

The DBMS command performs a multiple DBM. For example, to create a full graphics 0 screen, the following two commands must be performed:

```
  0 DBPTR     (new list)
24 2 DBMS     (24 graphic 0 lines)
```

This would create a full graphics 8 screen:

```
  0 DBPTR
192 15 DBMS     (Antic 15 = graphic 8)
```

(192 graphic 8 lines fill one video screen)

DBMS (cont'd)

Mixed lists are also possible:

```
    0 DBPTR
160 15 DBMS
  4  2 DBMS
```

This would create a screen of 160 graphic 8 lines with four text lines at the bottom.

DBIN (antic-mode block# ---)

Oftentimes, it is desirable to slightly change the existing display list to obtain special effects midway through a running program. The DBIN command allows insertion of new display blocks within the current display list. This command inserts "antic-mode" into the block specified by "block#". Whatever was in the block "block#" and following is pushed back one block. For example:

Display list

block #0	2
1	2
2	8

with the above display list, a

```
15  1  DBIN
```

would give the following display list.

Display list

block # 0	2
1	15
2	2
3	8

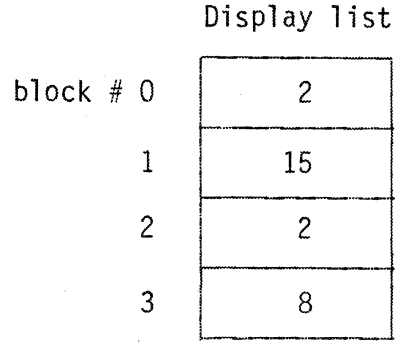
The DBIN allows the user to create new display lists without the need to duplicate already existing display list sections.

DBINS (#times antic-mode block# ---)

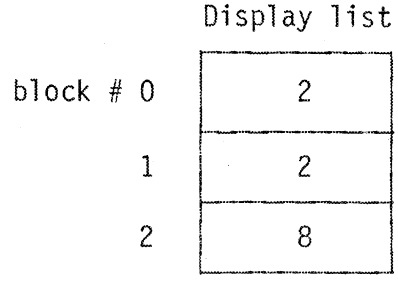
This command repeats "antic-mode block# DBIN" the specified number of times.

DBDEL (block# ---)

The DBDEL command serves as the logical complement to the DBIN command. Thus, after inserting a temporary display block, the DBDEL command may be used to delete that display block once it is no longer needed:



1 DBDEL would give:



Note: Deleting non-existing display blocks gives unexpected results.

DBDELS (#times block# ---)

This command performs "block# DBDEL" the specified number of times. This serves as the logical complement to the DBINS command.

DBDELL (---)

This form of the DBDEL command deletes the last display block created using the DBM command. For example:

	Display list
block # 0	2
1	15
2	2
3	8

DBDELL would give:

	Display list
block # 0	2
1	15
2	2

The main use for the DBDELL command is for "backing up" and re-entering a display block when an error has been made while creating a display list directly at the keyboard. The DBDELL command can be used successively for deleting a section of display blocks at the end of the current display list. There is no plural command for DBDELL command as its use is rather limited.

?ANTMOD (block# --- antic-mode)

Occasionally, it is desirable to know what antic-mode is being used for a particular display block (such as for a text output routine -- text should not be output on a hi-resolution line, for example). This command returns the antic-mode of the specified block.

DBMOD (modifier block# ---)

When creating display lists, it is possible to give extra meaning to a particular block or section of blocks in the list. This is accomplished by using one or more of the three available antic-modifiers: vertical scroll modifier (VRTMOD), horizontal scroll modifier (HRZMOD), and the display-list interrupt (INTMOD). The following are examples of each:

```
VRTMOD 0 DBMOD
HRZMOD 3 DBMOD
INTMOD 5 DBMOD
```

DBMOD (cont'd)

There are several methods in which to put more than one modifier on a given display block. For example, each of the following would give the same final result:

```
VRTMOD 20 DBMOD
HRZMOD 20 DBMOD
```

or

```
VRTMOD HRZMOD + 20 DBMOD
```

To attach all three modifiers, the best method is:

```
VRTMOD HRZMOD INTMOD + + 20 DBMOD
```

It should also be noted that it is possible to create modified display blocks, thus reducing the need for the DBMOD command:

```
HRZMOD 2 + DBM
```

This would create one graphic 0 line with a horizontal modifier. It is also easy to obtain 16 lines of hi-resolution graphics with both horizontal and vertical scroll modifiers:

```
16 VRTMOD HRZMOD 15 + + DBMS
```

CAUTION: VRTMOD and HRZMOD can only be used on antic-modes 2 through 15 ($2-$F).

(Note: There is one additional modifier, JMPMOD; however its use is absolutely forbidden! This has been defined as it will be implemented in the next release.)

DBMODL (modifier ---)

This command modifies the last display-block in the display list.

?DBMODS (block# --- modifiers)

This returns the modifiers on the specified display block. For example:

```
VRTMOD 2 + 0 DBM
0 ?DBMODS
```

would give VRTMOD. Also:

```
VRTMOD HRZMOD 2 + + 0 DBM
0 ?DBMODS
```

would give VRTMOD + HRZMOD. To test for VRTMOD, the following method must be used:

```
0 ?DBMODS
VRTMOD AND
```

The last line leaves only the vertical modifier, if present, or leaves 0 indicating no vertical modifier.

DBREM (block# ---)

The DBREM command removes all modifiers from the specified display block. Care should be taken when stripping modifiers, as stripping a horizontal modifier (if present) will change the size of the video memory.

DBREMS (#times block# ---)

This performs "block# DBREM" the specified number of times.

DBREML (---)

This removes the modifiers from the last display block in the current display list.

?DBVAL (block# --- info)

The ?DBVAL command returns all information about the display block specified, i.e., the antic mode and any modifiers. This information is returned as one value.

DBWID (width ---)

The DBWID command is used to set the desired playfield width so that the address array DBLST gives the proper values. Legal settings are: 1 - narrow, 2 - normal, and 3 - wide.

USRDSP (---)

Once a display list has been created, USRDSP activates the new list.

MIXED (---)

The MIXED command performs a USRDSP then instructs the Atari operating system to re-direct all output to the video display memory specified by the newly created display list.

DMPLST (---)

The DMPLST command instructs the display list assembler to give a complete, informative listing of the display list last created.

DBADR (block# --- address)

The DBADR command is one of the most useful commands to the programmer. Given a display block number, it returns the address of the first byte of that display block. This is extremely useful for determining where output text or graphic displays should be located.

DMCLR (---)

The DMCLR command clears the display memory pointed to by the display list currently being created. It clears to the top of memory.

In addition, there are various variables available to the programmer:

DSPEND — Points to the end of the current display list. It is an offset from 0 DSPLST.

DSPBLK — Contains the number of the next display block to be created.

DMLOC — Points to the beginning of display memory.

LSTLOC — Contains the address of where the display list is to reside in memory.

DBLST — Is an array of addresses used by DBADR.

DSPLST — Is a byte array containing the display list currently being created. DSPEND above points to the end of the list in this array.

```
Screen:  30
  0 ( Graph Sys:  tables                  )
  1 '( TRANSIENT TRANSIENT )(  )
  2 BASE @ DECIMAL
  3 '( CTABLE )( 45 KLOAD )
  4
  5 LABEL BLKNML
  6    0 C,   0 C, 40 C, 40 C, 40 C,
  7   40 C, 20 C, 20 C, 10 C, 10 C,
  8   20 C, 20 C, 20 C, 40 C, 40 C,
  9   40 C,
 10
 11 CTABLE HSOFS
 12   50 C,   4 C,   5 C, 50 C,
 13
 14 TABLE BLKOFS
 15   50 , -5 , 50 , 5 ,                ==>
```

```
Screen:  31
  0 ( Graph Sys:  variables               )
  1
  2 CTABLE BYTBLK 16 ALLOT
  3 BLKNML 0 BYTBLK 16 CMOVE
  4
  5  255   CARRAY DSPLST
  6  255    ARRAY DBLST
  7    5 VARIABLE HS#/
  8    0 VARIABLE DSPBLK
  9    0 VARIABLE PGECRS
 10    0 VARIABLE DSPEND
 11    0 VARIABLE NWLST
 12    6 VARIABLE SDTMP
 13    0 VARIABLE DBCNT
 14    0 VARIABLE DBVRT
 15                                    -->
```

```
Screen:  32
  0 ( Graph Sys:  constants               )
  1 HEX
  2    0 VARIABLE DMLOC
  3    0 VARIABLE LSTLOC
  4
  5   10 CONSTANT HRZMOD
  6   20 CONSTANT VRTMOD
  7   40 CONSTANT JMPMOD
  8   80 CONSTANT INTMOD
  9
 10 DECIMAL
 11
 12   11   CARRAY JMPDAT
 13   10   CARRAY JMPSTT
 14
 15   0 JMPDAT 11 ERASE                ==>
```

```
Screen:  33
  0 ( Graph Sys:  [DBINIT]  DBINIT )
  1
  2
  3 : (DBINIT)       ( DM  LIST  -- )
  4   LSTLOC !     DUP
  5   DMLOC !      0 DBLST  !
  6   0 DSPEND !   0 PGECRS !
  7   0 DSPBLK !   1 NWLST  !
  8   0 JMPDAT 11 ERASE ;
  9
 10 : DBINIT                      ( -- )
 11   106 C@ 256 * 7936 -
 12   DUP 256 -
 13   (DBINIT) ;
 14
 15 DBINIT                             -->
```

```
Screen:  34
  0 ( Graph Sys:  HINYB  EODB       )
  1 DECIMAL
  2
  3 : HINYB              ( nnnnn -- n )
  4   61440 AND 4096 / 16 + 15 AND ;
  5
  6 : EODB                  ( n -- a )
  7   31 AND DUP 15 AND   ( Find end)
  8   BYTBLK C@ SWAP      ( of disp )
  9   15 )                ( block )
 10   IF              ( Horz. scroll )
 11     DUP HS#/ @ / +
 12   ENDIF
 13   DSPBLK @        ( Update the  )
 14   DBLST @         ( addr list )
 15   + 1- ;                          ==>
```

```
Screen:  35
  0 ( Graph Sys:  DBPTR                   )
  1
  2 : DBPTR                 ( blk# -- )
  3   DMLOC @ 0 DBLST !
  4   0 PGECRS !      0 NWLST !
  5   DUP DSPBLK !    DUP DSPEND !
  6   0 JMPDAT C@ -DUP
  7   IF 1+ 1 DO
  8        I JMPDAT C@ OVER (
  9        IF
 10          2 DSPEND +!
 11        ELSE
 12          I 1- 0 JMPDAT C! LEAVE
 13        ENDIF
 14      LOOP
 15   ENDIF DROP                         -->
```

```
Screen:  36
  0 ( Graph Sys:  JMPINS  DBPTR     )
  1
  2   DSPEND @ DSPBLK @ =
  3   IF
  4     1 NWLST !
  5   ELSE
  6     DSPBLK @ DBLST @
  7     1- HINYB PGECRS !
  8   ENDIF ;
  9
 10 : JMPINS                  ( n -- )
 11   DUP JMPMOD OR
 12   DSPEND @ SWAP OVER
 13   DSPLST C! 1+ NWLST @
 14   IF
 15     0 NWLST !                  ==>
```

```
Screen:  37
  0 ( Graph Sys:  JMPINS            )
  1
  2     DMLOC @ DUP DUP 40 +
  3     HINYB SWAP HINYB <>
  4     IF
  5       HINYB 1+ 4096 *
  6     ENDIF
  7   ELSE
  8     PGECRS @ 4096 *
  9   ENDIF
 10   DUP ROT DSPLST !
 11   3 DSPEND +!
 12   DSPBLK @ 0 JMPDAT C@ 1+
 13   DUP 0 JMPDAT C!
 14   JMPDAT C! ;
 15                              -->
```

```
Screen:  38
  0 ( Graph Sys:  DBCRT             )
  1
  2 : DBCRT                    ( n -- )
  3   DUP EODB DUP
  4   HINYB PGECRS @
  5   OVER PGECRS ! <>
  6   IF
  7     DROP JMPINS
  8     DSPBLK @ DBLST ! EODB
  9   ELSE
 10     SWAP DSPEND @
 11     DSPLST C!
 12     1 DSPEND +!
 13   ENDIF
 14   1+ DSPBLK 1 OVER +!
 15   @ DBLST ! ;                ==>
```

```
Screen:  39
  0 ( Graph Sys:  DBM               )
  1
  2 : DBM                      ( n -- )
  3   DUP 15 AND
  4   IF              ( antic instr.? )
  5     191 AND       ( strip jump    )
  6     DBCRT
  7   ELSE              ( scan lines )
  8     DSPEND @
  9     DSPLST C!
 10     1 DSPEND +!
 11     DSPBLK @ DUP DBLST @
 12     SWAP 1+ DBLST !
 13     1 DSPBLK +!
 14   ENDIF ;
 15                              -->
```

```
Screen:  40
  0 ( Graph Sys:  LSTSV             )
  1
  2 : LSTSV              ( blk# -- a )
  3
  4   DSPEND @           ( pt to blk )
  5   DSPLST 65 OVER C!
  6   SWAP DBPTR
  7
  8   DSPEND @ DSPLST    ( save list )
  9   DUP SDTMP @ + ROT
 10   >R OVER R> -
 11   ABS 1+ <CMOVE
 12
 13   DSPEND @ DSPLST    ( leave save)
 14   SDTMP @ + ;        ( address )
 15                              ==>
```

```
Screen:  41
  0 ( Graph Sys:  LSTRST            )
  1
  2 : LSTRST                   ( a -- )
  3   BEGIN
  4     DUP C@ DUP 65 <>     ( eolst?)
  5   WHILE
  6     DUP 15 AND 0#
  7     OVER 64 AND 0# AND
  8     IF                   ( jump? )
  9       191 AND DBM 3 +
 10     ELSE                 ( norma )
 11       DBM 1+
 12     ENDIF
 13   REPEAT
 14   2DROP ;
 15                              -->
```

```
Screen:  42
  0 ( Graph Sys:  DBIN DBDEL DBMOD )
  1
  2 : DBIN                    ( n n -- )
  3   LSTSV SWAP DBM LSTRST ;
  4
  5 : DBDEL                   ( n -- )
  6   DUP 1+ LSTSV SWAP
  7   DBPTR LSTRST ;
  8
  9 : DBMOD                   ( n n -- )
 10   DUP 1+ LSTSV
 11   SWAP DBPTR SWAP
 12   DSPEND @ DSPLST C@ OR
 13   DBM LSTRST ;
 14
 15                                  ==>
```

```
Screen:  43
  0 ( DBREM  DBDELL  DBMODL DBREML )
  1
  2 : DBREM                   ( n -- )
  3   DUP 1+ LSTSV
  4   SWAP DBPTR
  5   DSPEND @ DSPLST C@
  6   15 AND DBM LSTRST ;
  7
  8 : DBDELL                    ( -- )
  9   DSPBLK @ 1- DBPTR ;
 10
 11 : DBMODL                    ( -- )
 12   DSPBLK @ 1- DBMOD ;
 13
 14 : DBREML                    ( -- )
 15   DSPBLK @ 1- DBREM ;            -->
```

```
Screen:  44
  0 ( Graph Sys:  ?DBVAL              )
  1
  2 : ?DBVAL                  ( n -- )
  3   DSPBLK C@ 0 JMPDAT C@
  4   ROT DBPTR
  5   DSPEND @ DSPLST C@
  6   <ROT 0 JMPDAT C!
  7   DBPTR ;
  8
  9
 10
 11
 12
 13
 14
 15                                  ==>
```

```
Screen:  45
  0 ( Graph Sys:  ?ANTMOD  ?DBMODS )
  1
  2 : ?ANTMOD                   ( n -- )
  3   ?DBVAL DUP 15 AND 0=
  4   IF 127 ELSE 15 ENDIF
  5   AND ;
  6
  7 : ?DBMODS                   ( n -- )
  8   ?DBVAL DUP 15 AND 0=
  9   IF
 10     DROP 0
 11   ELSE
 12     240 AND
 13   ENDIF ;
 14
 15                                  -->
```

```
Screen:  46
  0 ( Graph Sys:  DBMS  DBDELS      )
  1
  2 : DBMS                    ( # n -- )
  3   SWAP 0
  4   DO
  5     DUP DBM
  6   LOOP
  7   DROP ;
  8
  9 : DBDELS                  ( # n -- )
 10   SWAP 0
 11   DO
 12     DUP DBDEL
 13   LOOP
 14   DROP ;
 15                                  ==>
```

```
Screen:  47
  0 ( Graph Sys:  DBREMS            )
  1
  2 : DBREMS                  ( # n -- )
  3   SWAP 0
  4   DO
  5     DUP DBREM 1+
  6   LOOP
  7   DROP ;
  8
  9
 10
 11
 12
 13
 14
 15                                  -->
```

```
Screen:  48
  0 ( Graph Sys:  DBINS                 )
  1
  2 : DBINS                ( # n n -- )
  3   ROT DUP DBCNT !
  4   6 + SDTMP @ SWAP SDTMP !
  5   (ROT DUP LSTSV
  6   SWAP DBPTR SWAP
  7   BEGIN
  8     DBCNT @       ( count zero? )
  9   WHILE
 10     DUP DBM      ( keep creating)
 11     -1 DBCNT +!  ( dec. counter )
 12   REPEAT
 13   DROP LSTRST    ( unsave list )
 14   SDTMP ! ;
 15                                ==>
```

```
Screen:  49
  0 ( Graph Sys:  SCRWID  LSTMV     )
  1
  2 : SCRWID               ( width -- )
  3   559 C@ 252 AND
  4   OR 559 C! ;
  5
  6 : LSTMV                      ( -- )
  7   LSTLOC @ DSPEND @
  8   DSPLST 65
  9   OVER C! 1+ !
 10   0 DSPLST LSTLOC @
 11   DSPEND @ CMOVE ;
 12
 13
 14
 15                                -->
```

```
Screen:  50
  0 ( Graph Sys:  USRDSP MIXED      )
  1
  2 : USRDSP                     ( -- )
  3   LSTMV
  4   559 C@ 3 AND
  5   0 SCRWID
  6   LSTLOC @ 560 !
  7   SCRWID  ;
  8
  9 : MIXED                      ( -- )
 10   DMLOC @ 88 !
 11   USRDSP ;
 12
 13
 14
 15                                ==>
```

```
Screen:  51
  0 ( Graph Sys:  DMPLST
  1
  2 : DMPLST                          ( --
  3   CR DSPBLK @ -DUP
  4   IF
  5     CR ."  BLK     ADDR"
  6     ."     BYTE    MODS"
  7     CR ."  ---     ----"
  8     ."     ----    ----"
  9     0 DO
 10       CR I 3 .R
 11       I DBLST @ 9 U.R
 12       I ?ANTMOD  8 U.R
 13       I ?DBMODS  -DUP
 14       IF
 15         3 SPACES                 -->
```

```
Screen:  52
  0 ( Graph Sys:  DMPLST
  1
  2          DUP 128 AND
  3          IF ." I" ENDIF
  4          DUP 32 AND
  5          IF ." V" ENDIF
  6          DUP 16 AND
  7          IF ." H" ENDIF
  8              64 AND
  9          IF ." J " I DBLST @ U.
 10          ENDIF
 11        ENDIF ?EXIT
 12      LOOP
 13    ELSE
 14      ." No display list"
 15    ENDIF CR ;                    ==>
```

```
Screen:  53
  0 ( Graph Sys: DMCHG DMCLR DBADR
  1
  2 : DMCLR                           ( --
  3   DMLOC @
  4   106 @ 256 *
  5   OVER -
  6   ERASE ;
  7
  8 : DBADR                  ( blk# -- a
  9   DBLST @ ;
 10
 11
 12
 13
 14
 15                                -->
```

```
Screen:  54
  0 ( VAL-FORTH GRAPHIC SYSTEM 1.1 )
  1
  2 : DBWID                ( width -- )
  3   0 LSTSV SWAP BLKNML
  4   [ 0 BYTBLK ] LITERAL
  5   16 CMOVE
  6   BLKOFS @ [ 0 BYTBLK 16
  7   OVER + ] LITERAL LITERAL
  8   DO
  9     I C@ DUP 3 PICK / + I C!
 10   1 /LOOP
 11   HSOFS C@ HS#/ !
 12   LSTRST ;
 13
 14 '( PERMANENT PERMANENT )(   )
 15 BASE !
```

```
Screen:  55
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  56
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  57
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  58
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  59
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  60
  0 ( Transients:  setup                )
  1 BASE @ DCX
  2
  3 HERE
  4
  5
  6 741 @ 4000 - DP !
  7 ( SUGGESTED PLACEMENT OF TAREA )
  8
  9
 10 HERE CONSTANT TAREA
 11    0 VARIABLE TP
 12    1 VARIABLE TPFLAG
 13      VARIABLE OLDDP
 14
 15                                  ==>
```

```
Screen:  61
  0 ( Xsients: TRANSIENT PERMANENT )
  1
  2 : TRANSIENT                 ( -- )
  3   TPFLAG @ NOT
  4   IF HERE OLDDP ! TP @ DP !
  5      1 TPFLAG !
  6   ENDIF ;
  7
  8 : PERMANENT                 ( -- )
  9   TPFLAG @
 10   IF HERE TP ! OLDDP @ DP !
 11      0 TPFLAG !
 12   ENDIF ;
 13
 14
 15                                  -->
```

```
Screen:  62
  0 ( Transients:  DISPOSE          )
  1 : DISPOSE   PERMANENT
  2   CR ." Disposing..." VOC-LINK
  3   BEGIN DUP    0 53279 C!
  4    BEGIN @ DUP TAREA U<
  5    UNTIL DUP ROT ! DUP 0=
  6   UNTIL DROP VOC-LINK @
  7   BEGIN DUP 4 -
  8    BEGIN DUP    0 53279 C!
  9     BEGIN PFA LFA @ DUP TAREA U<
 10     UNTIL
 11      DUP ROT PFA LFA ! DUP 0=
 12    UNTIL DROP @ DUP 0=
 13   UNTIL DROP [COMPILE] FORTH
 14   DEFINITIONS ." Done" CR ;
 15   PERMANENT              BASE !
```

```
Screen:  63
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  64
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  65
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  90
 0 ( Utils:  CARRAY  ARRAY          )
 1 BASE @ HEX
 2 : CARRAY          ( cccc, n -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4     ALLOT
 5   ;CODE CA C, CA C, 18 C,
 6   A5 C, W C,  69 C, 02 C, 95 C,
 7   00 C, 98 C, 65 C, W 1+ C,
 8   95 C, 01 C, 4C C,
 9   ' + ( CFA @ ) , C;
10
11 : ARRAY          ( cccc, n -- )
12   CREATE SMUDGE ( cccc: n -- a )
13     2* ALLOT
14   ;CODE 16 C, 00 C, 36 C, 01 C,
15    4C C, ' CARRAY 08 + , C;  ==>
```

```
Screen:  91
 0 ( Utils:  CTABLE  TABLE          )
 1
 2 : CTABLE          ( cccc, -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4   ;CODE
 5   4C C, ' CARRAY 08 + , C;
 6
 7 : TABLE           ( cccc, -- )
 8   CREATE SMUDGE ( cccc: n -- a )
 9   ;CODE
10   4C C, ' ARRAY 0A + , C;
11
12
13
14
15                                 -->
```

```
Screen:  92
 0 ( Utils:  2CARRAY  2ARRAY        )
 1
 2 : 2CARRAY       ( cccc, n n -- )
 3   <BUILDS       ( cccc: n n -- a )
 4    SWAP DUP , * ALLOT
 5   DOES>
 6     DUP >R @ * + R> + 2+ ;
 7
 8 : 2ARRAY        ( cccc, n n -- )
 9   <BUILDS       ( cccc: n n -- a )
10    SWAP DUP , * 2* ALLOT
11   DOES>
12     DUP >R @ * + 2* R> + 2+ ;
13
14
15                                 ==>
```

```
Screen:  93
 0 ( Utils:  XC!  X!                )
 1
 2 : XC!     ( n0...nm cnt addr -- )
 3   OVER 1- + >R 0
 4   DO J I - C!
 5   LOOP R> DROP ;
 6
 7 : X!      ( n0...nm cnt addr -- )
 8   OVER 1- 2* + >R 0
 9   DO J I 2* - !
10   LOOP R> DROP ;
11
12 ( Caution: Remember limitation
13 ( on stack size of 30 values
14 ( because of OS conflict. )
15                                 -->
```

```
Screen:  94
 0 ( Utils:  CVECTOR  VECTOR        )
 1
 2 : CVECTOR         ( cccc, cnt -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4   HERE OVER ALLOT XC!
 5   ;CODE
 6   4C C, ' CARRAY 08 + , C;
 7
 8 : VECTOR          ( cccc, cnt -- )
 9   CREATE SMUDGE ( cccc: n -- a )
10   HERE OVER 2* ALLOT X!
11   ;CODE
12   4C C, ' ARRAY  0A + , C;
13
14
15                          BASE !
```

```
Screen:  95
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screens 96 thru 167 are blank

```
Screen: 168
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 169
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 170
  0 CONTENTS OF THIS DISK:
  1
  2 DISPLAY FORMATTER:        30 LOAD
  3 TRANSIENTS:               60 LOAD
  4 ARRAYS & THEIR COUSINS:   90 LOAD
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 171
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 172
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 173
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 174
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 175
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 176
 0 ( Error messages                                              )
 1
 2 Stack empty
 3
 4 Dictionary full
 5
 6 Wrong addressing mode
 7
 8 Is not unique
 9
10 Value error
11
12 Disk address error
13
14 Stack full
15
```

```
Screen: 177
 0 Disk Error!
 1
 2 Dictionary too big
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 178
 0 ( Error messages                                              )
 1
 2 Use only in Definitions
 3
 4 Execution only
 5
 6 Conditionals not paired
 7
 8 Definition not finished
 9
10 In protected dictionary
11
12 Use only when loading
13
14 Off current screen
15
```

```
Screen: 179
 0 Declare VOCABULARY
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

HANDY REFERENCE CARD

valFORTH T.M.

SOFTWARE SYSTEM

DISPLAY FORMATTER

(DBINIT)	(dmem dlist ---)	This command initializes the display formatter using "dlist" as the target address for the display list, and "dmem" as the target address for display memory.
DBINIT	(---)	This command initializes the display formatter setting the display memory address to top-of-memory minus $1F00. The display list is targeted for $100 bytes below the display memory.
DBM	(opcode ---)	The DBM command adds "opcode" to the end of the current display list.
DBMS	(#times opcode ---)	The DBMS command performs a multiple DBM command as described above.
DBPTR	(block# ---)	This command makes the specified block the next block to be created with the DBM command. It essentially makes block#-1 the end of the current list.
DBIN	(opcode block# ---)	The DBIN command inserts the specified opcode into the specified block. That block and all following blocks are pushed back one block.
DBINS	(#times opcode block# ---)	This command performs a multiple DBIN command as described above.
DBDEL	(block # ---)	The DBDEL command deletes the specified block from the current display list.
DBDELS	(#times block# ---)	This command performs a multiple DBDEL command as described above.
DBDELL	(---)	The DBDELL command deletes the last block of the current display list.
DBMOD	(modifier block# ---)	This command modifies the specified block. Legal modifiers are VRTMOD, HRZMOD, and INTMOD.
DBMODL	(modifier ---)	This command modifies the last block of the current display list.
DBREM	(block# ---)	The DBREM command removes all modifiers from the specified block. Note that if a HRZMOD is stripped, display memory allocation will change.
DBREMS	(#times block# ---)	This command is a multiple DBREM command.
DBREML	(---)	This command strips modifiers from the last block of the current display list.
?DBVAL	(block# --- value)	The ?DBVAL command returns all information about the display block specified, i.e., the antic mode and any modifiers. This information is returned as one value.
?ANTMOD	(block# --- antic-mode)	This command returns the antic-mode (or opcode) of the specified block.
?DBMODS	(block# --- modifiers)	The ?DBMODS command returns the modifiers for the specified block. This information is returned as one value. See documentation for notes on interpretation of this value.
DBWID	(width ---)	The DBWID command sets the display formatter up for narrow (1), normal (2), or wide (3) screen display.
DBADR	(block# --- address)	Given a display block number, it returns the address of the first byte of that display block.
DMCLR	(---)	The DMCLR command clears the display memory pointed to by the display list currently being created.
USRDSP	(---)	Once a display list has been created, USRDSP activates the new list.
MIXED	(---)	The MIXED command performs a USRDSP then instructs the Atari operating system to re-direct all output to the video display memory specified by the newly created display list.
DMPLST	(---)	The DMPLST command gives a complete, informative listing of the display list currently being created.
DSPEND	(--- address)	A variable which contains a pointer to the end of the current display list. It is an offset from 0 DSPLST.
DSPBLK	(--- address)	A variable containing the number of the next display block to be created.
DMLOC	(--- address)	A variable which contains the target address of the display memory pointed to by the current display list.
LSTLOC	(--- address)	A variable which contains the target address for the current display list.
DBLST	(block# --- address)	An array of addresses used by DBADR.
DSPLST	(pointer --- address)	A c-array containing the display list currently being created. DSPEND above points to the end of the list in this array.

HANDY REFERENCE CARD

valFORTH T.M.

SOFTWARE SYSTEM

DISPLAY FORMATTER

The Character Modes

There are 6 character modes (opcodes 2 thru 7). All character modes work in the same way, i.e., the values in display memory are indices to a large "n" by 8 byte array. In some of these modes, the highest one or two bits are used to specify a color with only the remaining lower bits used for indexing. The following table gives information about each of the modes:

Antic mode	2	3	4	5	6	7
Basic mode	0	---	---	---	1	2
# color *	1.5	1.5	5	5	5	5
Chars/line narrow wid	32	32	32	32	16	16
Chars/line normal wid	40	40	40	40	20	20
Chars/line wide screen	48	48	48	48	24	24
Scan lines/ pixel	8	10	8	16	8	16
Bits/pixel	1	1	2	2	1	1
Color clocks per pixel	.5	.5	1	1	1	1

Colors

mode 2: Takes the color of PF2 with the lum of PF1
(Artifacting/bleed very noticeable)
mode 3: Same as above
mode 4: Two bits/pixel in character definitions
00 = BAK 01 = PF0 10 = PF1
11 = PF2 if bit 7 of index = 0, else PF3
mode 5: Same as 4 above
mode 6: Most significant two bits of index
0 = PF0 1 = PF1 etc.
mode 7: Same as 6 above

The Graphic Modes

There are 8 graphic modes. Unlike character modes, the values in display memory are not indices into an array of character definitions, but rather are the definitions themselves. Depending on the graphic mode, these values give different results. The following table gives various information about each mode.

Antic mode	8	9	A	B	C	D	E	F*
Basic mode	3	4	5	6	---	7	---	8
# colors	4	2	4	2	2	4	4	1.5
bytes/line narrow wid	8	8	16	16	16	32	32	32
bytes/line normal wid	10	10	20	20	20	40	40	40
bytes/line wide screen	12	12	24	24	24	48	48	48
Pixels per normal wid	40	80	80	160	160	160	160	320
Scan lines/pixel	8	4	4	2	1	2	1	1
Bits/pixel	2	1	2	1	1	2	2	1
Color clocks per pixel	4	2	2	1	1	1	1	.5

*Mode F values differ when in GTIA modes

Colors

mode 8: Two bits/pixel, 4 pixels/byte
00 = BAK 01 = PF0 10 = PF1 11 = PF2
mode 9: One bit/pixel, 8 pixels/byte
0 = BAK 1 = PF0
mode A: Same as mode 8 above
mode B: Same as mode 9 above
mode C: Same as mode 9 above
mode D: Same as mode 8 above
mode E: Same as mode 8 above
mode F: Take the color of PF2 and lum of PF1
(if not in a GTIA mode)